Docker Nextcloud Image selbst bauen

Möchte man ein Nextcloud-Docker-Image anpassen oder selbst bauen, sind folgende Schritte notwendig.
Beispiel: Nextcloud 33.0.5 mit Apache-Image.

Ich benutze die Apache Version

Basis-Image

Basis-Image holen

```
# Quell-Repo klonen
git clone https://github.com/nextcloud/docker.git
cd docker/33/apache
```

Update daes Basisimages:

in das Docker Directory wechseln und Image updaten mit:

```
git pull
```

2. prüfen, ob die gewünschte Version im Dockerfile vorhanden ist, in unserem Beispiel ENV NEXTCLOUD_VERSION 32.0.4

Dockerfile eventuell anpassen

3. Im Dockerfile kann man einige Variable anpassen, wenn man das möchte. Beispiel:

```
ENV PHP_MEMORY_LIMIT 2048M
ENV PHP_UPLOAD_LIMIT 2048M
ENV PHP_OPCACHE_MEMORY_CONSUMPTION 256
opcache.interned_strings_buffer=64
```

3.1. Besser ist es jedoch, ein eigenes volumne in docker-compose einzubinden. Beispiel

Beispielordnerstruktur:

```
projektordner/
├── docker-compose.yml
├── my-php/
│   └── 99-custom.ini   ← Hier liegt deine PHP-Konfig
```

Beispiel für eine 99-custom.ini

```
; -------------------------------------
```

```
; Benutzerdefinierte PHP-Einstellungen
; ------------------------------------

; Arbeitsspeicher
memory_limit = 2048M

; Upload-Limit für einzelne Dateien
upload_max_filesize = 2048M

; Maximalgröße von POST-Daten
post_max_size = 512M

; Maximale Skriptlaufzeit
max_execution_time = 300
max_input_time = 300

; Maximale Anzahl gleichzeitiger Datei-Uploads
max_file_uploads = 200

; Zeitzone (optional)
date.timezone = Europe/Vienna
```

Testbeispiel, ob die Werte auch wirklich geladen werden:

```
docker exec -it nextcloud php -i | grep memory_limit
docker exec -it nextcloud php -i | grep upload_max_filesize
```

So bindest du die Datei in docker-compose ein.

```
- ./my-php/99-custom.ini:/usr/local/etc/php/conf.d/99-custom.ini
```

docker-compose.yml anpassen und build starten

4. Ich lasse docker-compose.yml den build selbst machen. Dazu verändert man folgende Zeilen in docker-compose.yml in derm Abschnitt von Nextcloud Image

```
#---- habe ein eigenes Image erstellt --------
 build:
    context: /home-to-own-docker-image/docker/31/apache
    dockerfile: Dockerfile
 image: nextcloud:33.0.5-apache
```

5. Build Prozess starten

- zuerst Syntax überprüfen

```
docker compose config
```

```
docker compose up -d
```

Ist ein lokales Image vorhanden, wird der build Prozess nicht gestartet. Das kann man erzwingen:

```
docker compose up -d --build
```

6. Abwarten, bis der build Prozess durchgelaufen ist, danach kann man die Version prüfen.

Mehrere Instanzen in einem Docker Daemon

Laufen alle Nextcloud Container am selben Docker daemon, reicht es völlig, **ein einziges Mal ein Tagged Image zu haben**. Jede `docker-compose.yml` muss dann nur noch **dieses Tag als `image:` verwenden** – ohne `build:` -Block. Der Daemon liefert das Image aus seinem lokalen Cache; es wird **nichts neu gebaut und nichts gepullt**

Vorgangsweise:

1. Image einmalig taggen

```
docker tag $(docker images -q --filter reference='nextcloud:33.0.5-apache' |
head -n1) my-nextcloud:33.0.5-apache
```

2. setze in alle docker-compose.yml folgende ein (ohnue build:)

```
image: my-nextcloud:33.0.5-apache
```

3. Container pro Nextcloud Instanz neu erzeugen

```
docker compose pull   # wird sofort übersprungen, weil Image lokal
docker compose up -d  # startet mit dem vorhandenen Layer-Cache
```

Tipp: Versionsvariable nur 1x pflegen:

1. Lege in jedem Instanz-Verzeichnis (oder als Symlink auf eine gemeinsame Datei) eine **`.env`** an – Compose ersetzt Variablen automatisch

```
# .env
NEXTCLOUD_IMAGE=my-nextcloud:33.0.5-apache
```

2. In jedem docker-compose.yml im Nextcloud Abschnitt:

```
image: ${NEXTCLOUD_IMAGE}
```

2. Baue das neue Image **einmal**, ändere nur die Zeile in **einer** `.env`, , wen du einen Symlink gesetzt hast, sonst in jeder, dann:

```
docker compose up -d # Container aller Instanzen werden ersetzt
```

2. gemeinsame .env erzeugen

```
mkdir -p /srv/nextcloud/common
cat>/srv/nextcloud/common/.env <<'EOF'
NEXTCLOUD_IMAGE=my-nextcloud:32.0.4-apache
REDIS_HOST=redis
EOF
```

3. Symlink in jeder Instanz anlegen **(am beseten mit absolutem Pfad)**

```
ln -s /srv/nextcloud/common/.env .env
```

4. Docker-compose.yml nutzt die Variable

```
version: "3.9"
services:
  nextcloud:
    image: ${NEXTCLOUD_IMAGE}      # kommt aus der (verlinkten) .env
    # …
```

Docker Image auf neuen Server übertragen

1. Image nur einmal als Datei erzeugen

```
docker save my-nextcloud:32.0.4-apache \
  | gzip> my-nextcloud_32.0.4-apache.tar.gz
```

2. **Datei auf den Ziel-Host übertragen** (SCP, rsync, USB-Stick …)

```
scp -P Portnummer my-nextcloud_32.0.4-apache.tar.gz user@host2:/tmp/
```

3. dort einspielen

```
gunzip -c /tmp/my-nextcloud_32.0.4-apache.tar.gz | docker load
```

4. Prüfen, ob das Image vorhanden ist

```
docker images my-nextcloud:32.0.4-apache
```

Jetzt taucht es lokal als `my-nextcloud:32.0.4-apache` auf – Compose kann’s direkt nutzen.

5 Docker-compose kann das File direkt nutzen

```
services:
  nextcloud:
    image: my-nextcloud:32.0.4-apache
```

```
    pull_policy: never    # verhindert versehentliche Pull-Versuche
```

6. aufrufen mit

```
docker compose up -d      # reicht vollkommen, kein Pull nötig
```

Falls du ganz sicher gehen willst:

```
docker compose up -d --pull never --no-build
```

Redis (oder einzelne Services) updaten – pro Compose-Instanz

Im jeweiligen Compose-Projektordner (dort wo du `docker-compose.yml` /Symlink aufrufst):

```
docker compose pull redis
docker compose up -d --no-deps redis
```

- `pull redis` lädt das neue Image (falls Tag/Digest neuer ist)
- `up -d –no-deps redis` erstellt **nur Redis** neu, ohne alles andere neu zu starten Wenn du mehrere Services updaten willst:

```
docker compose pull redis mariadb
docker compose up -d --no-deps redis mariadb
```